Serviço de chat multi-usuário com tranferência de arquivos

Gilberto Kreisler

Curso de Bacharelado em Ciência da Computação
Universidade Federal de Pelotas (UFPel)

gkfneto@inf.ufpel.edu.br

Resumo. *Este meta-artigo descreve o protocolo implementado de uma aplicação de chat multi-usuário com suporte para a transferência de arquivo utilizando a linguagem Python 3. Para isso foi utilizado a biblioteca Sockets e Multithreading.*

1. Introdução

O projeto desenvolvido implementa um chat multi-usuário onde estes poderão trocar arquivo entre os usuários.

Neste projeto usaremos a linguagem de programação Python 3 junto com sua API *Socket* para fazer a comunicação entre cliente e servidor. Foi utilizado o protocolo TCP, pois precisamos da garantia da entrega das mensagens, bem como de sua integridade.

Para suportar a requisição de múltiplos usuários foi utilizada a biblioteca *Threading*. Portanto quando um novo usuário solicita conexão, após as verificações necessárias é criada uma nova *thread* para administrar a comunicação com o usuário enquanto a *thread* principal volta a aguardar novos usuário.

No lado do cliente são criadas duas *threads,* uma responsável por fazer o parse e organizar as mensagem enviadas pelo cliente. A outra é responsável por atender as mensagens recebidas do servidor. Portanto, uma *thread* apenas envia mensagens e a outra apenas recebe.

A aplicação não oferece uma UI complexa, sendo toda a interação através do terminal após executar a aplicação, também através do terminal.

Não foi pensado em padrões de segurança, pois o único foco é a transferência de mensagem entre usuários corretamente através do protocolo criado.

Quando um arquivo é enviado por um usuário, todos os outros conectados ao servidor recebem o arquivo obrigatoriamente.

2. Manual de uso

Para a utilização tanto da aplicação do servidor quanto da aplicação do cliente é necessário ter *Python 3.X* instalado.

As aplicações rodam diretamente através do terminal, dessa forma para executar os programas é necessário estar dentro do diretório onde está o programa e executar: *python3 programa.py*

O servidor não exige de nenhuma configuração adicional e já fica aguardando novas conexões logo após ser iniciado.

Quando o cliente é executado um diretório é criado dentro do */home* do usuário para o recebimento de arquivos. Depois é perguntado por um *nick*, sem um não é possível entrar na sala. A conexão só é estabelecida quando o usuário entra com um *nick* válido e único no chat.

A aplicação foi testada apenas no Ubuntu.

3. Implementação

A implementação do sistema de comunicação foi feita com a biblioteca *Sockets*, que oferece suporte para a comunicação. Foi utilizado o protocolo TCP pela necessidade da garantia da entrega dos pacotes. Para isso é necessário criar um socket tanto no servidor quanto no cliente, ambos configurados sob o mesmo endereço IP e porta.

Em uma tentativa de conexão mal sucedida, o socket gerado no cliente é descartado e um novo é criado, pois o servidor não aceita conexões de sockets que tiveram algum problema anteriormente.

Toda comunicação é iniciado pelo envio de códigos em um cabeçalho de tamanho fixo que contém informações sobre o tipo da mensagem e tamanho, para assim receber os dados da mensagem.

Na aplicação de usuário possui duas threads, uma apenas envia mensagens ao servidor e a outra apenas recebe. Portanto, uma comunicação é iniciada na thread que apenas envia só termina na thread que apenas recebe, assim a que recebe deve continuar de onde parou a comunicação.

Foi optado dessa forma, pois duas threads que escutam o mesmo buffer associado ao socket quebra o protocolo de comunicação, podendo ocorrer de uma thread ficar aguardando uma resposta do servidor, mas quem recebe é a outra.

O servidor só permite a conexão do usuário quando o nick do mesmo é único. Esses dados são mantidos em uma lista que associa o nick do usuário com o socket, pois cada usuário possui um valor único. Tendo garantido que o nick é válido, uma thread é criada para atender às requisições do cliente enquanto a principal volta a aguardar pelo pedido de conexão de novos clientes.

Comandos como */help* e */clear* são executados diretamente no usuário.

Tabela 1. Tabela de comandos

Comando	Descrição	Parâmetro
/list	Lista todos os usuários conectados	-
/help	Lista os comandos disponíveis	-
/clean	Limpa a janela do usuário	-

/quit	Desconecta do server	-
/nick	Trocar o apelido do usuário	novo_nick
/sendFile	Enviar arquivo	arquivo.ext

3.1. Mensagens de comunicação

O cabeçalho é um vetor de 10 posições obrigatoriamente, mesmo que o conteúdo seja menor. Na descrição da comunicação será usado *chaves {}* para representar o cabeçalho e *colchetes []* para representar a mensagem em si.

Tabela de códigos e significados:

Código	Significado
ACC	Aceito
list	listar usuário
nick	pedido de troca de nick
file	envio de arquivo
NE	nick existente
NC	nick trocado
msg	mensagem de texto
quit	terminar conexão

3.1.1. Conexão com o servidor

Cliente	Servidor - Aceito	Servidor - Negado
{sizeOf(nick)} + [nick]	{ACC}	{NE}

3.1.2. Listar usuários conectados (/list)

Cliente	Servidor
{list}	{sizeOf(userList)} + [userList]

3.1.3. Trocar nick (/nick novo_nick)

Cliente	Servidor - Aceito	Servidor - Negado
{nick sizeOf(nick)} + [nick]	{NC}	{NE}

3.1.4. Enviar arquivo (/sendFile nove_arquivo.ext)

Cliente
{file sizeOf(nome_arquivo)} + [nome_arquivo] + {sizeOf(file)} + [file_data]

3.1.5. Desconectar (/quit)

Cliente
{quit}

3.1.6. Enviar mensagem

Cliente
{msg sizeOf(text)} + [text]

3.1.7. Server BroadCast de Mensagens

3.1.7.1. Arquivos

Server
{file sizeOf(nick_sender) sizeOf(nome_arquivo)} + [nick_sender] + [nome_arquivo] + {sizeOf(file)} + [file_data]

3.1.7.2. Mensagens

Server
{msg sizeOf(nick_sender) sizeOf(text)} + [nick_sender] + [text]

4. Máquinas de estado

Esta seção apresenta as máquinas de estados possíveis para o servidor e para o cliente nesta aplicação.

4.1. Servidor

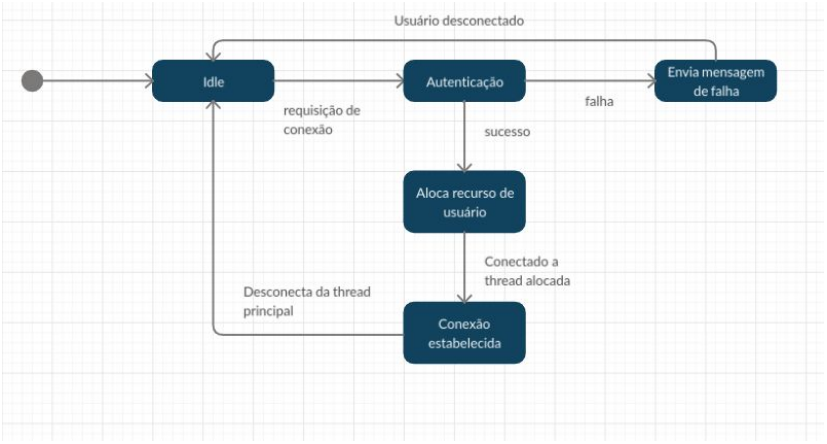

Figura 1. Thread principal do servidor

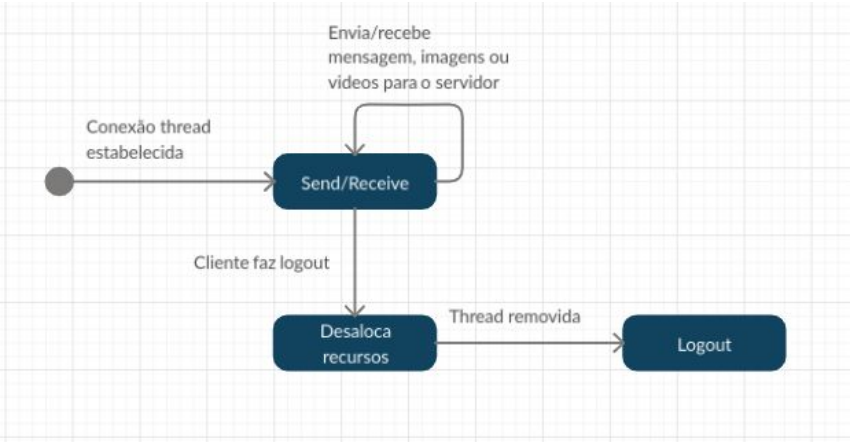

Figura 2. Thread dedicada a usuário

4.2. Cliente

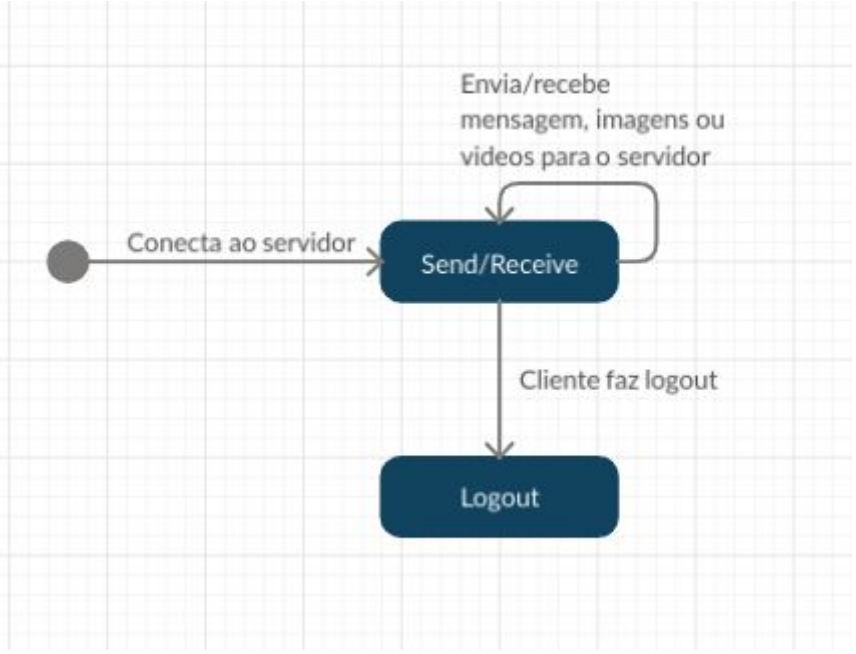

Figura 3. Máquina de estado do cliente

Referências

Basic IRC commands. Disponível em:
https://www.mirc.com/help/html/index.html?basic_irc_commands.html

Tutorial chatroom com python:

https://pythonprogramming.net/client-chatroom-sockets-tutorial-python-3/